A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids

V. Venkatakrishnan[1], H.D. Simon[1], and T. J. Barth[2]

Report RNR-91-024, September 1991

National Aeronautics and Space Administration

Ames Research Center
Moffett Field, California 94035

ARC 275 (Rev Feb 81)

A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids

V. Venkatakrishnan[1], H.D. Simon[1], and T. J. Barth[2]

Report RNR-91-024, September 1991

NAS Systems Division
Applied Research Branch
NASA Ames Research Center, Mail Stop T045-1
Moffett Field, CA 94035

September 9, 1991

Abstract. A mesh-vertex finite volume scheme for solving the Euler equations on triangular unstructured meshes is implemented on an MIMD (multiple instruction multiple data stream) parallel computer. Various partitioning strategies for distributing the work load on to the processors are discussed. Issues pertaining to the communication costs are also addressed. Finally, the performance of this unstructured computation on the Intel iPSC/860 is compared with that of a one processor of a Cray Y-MP, and with an earlier implementation.

(submitted to The Journal of Supercomputing)

---

[1]Applied Research Branch, Mail Stop T045-1, Numerical Aerodynamic Simulation (NAS) Systems Division, NASA Ames Research Center, Moffett Field, CA 94035. The author is an employee of Computer Sciences Corporation. This work was funded under contract NAS 2-12961.

[2]Computational Fluid Dynamics Branch, Mail Stop 202A, NASA Ames Research Center, Moffett Field, CA 94035.

A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids

V. Venkatakrishnan[1], Horst D. Simon[1],
and Timothy J. Barth[2]

Abstract

A mesh-vertex finite volume scheme for solving the Euler equations on triangular unstructured meshes is implemented on an MIMD (multiple instruction multiple data stream) parallel computer. Various partitioning strategies for distributing the work load on to the processors are discussed. Issues pertaining to the communication costs are also addressed. Finally, the performance of this unstructured computation on the Intel iPSC/860 is compared with that of a one processor Cray Y-MP, and with an earlier implementation on the Connection Machine.

1 Introduction

There has been a surge of activity in the area of parallel computing in the last few years. Parallel computers are becoming more accessible and many significant application programs have been successfully implemented. Many of the applications to date, however, have dealt with structured meshes which have a good deal of regularity in accessing data. This also facilitates the partitioning of the data and of the computation.

Triangular meshes have become quite popular in computational fluid dynamics. They are ideally suited for handling complex geometries and for adapting to flow features, such as shocks and boundary layers. Considerable attention has been focussed on improving the spatial operator, which has evolved to a very high degree of sophistication for the Euler and Navier-Stokes equations. Parallel computers seem to offer an avenue for doing large problems very fast due to their scalability. Before this goal can be realized, however, many fundamental issues need to be addressed. In the case of unstructured mesh computations, some of the issues that arise are the partitioning of the data and the computation among processors, communication at the inter-partition boundaries and dealing with global data addresses. Massively parallel computers have evolved to a degree that we feel it is the appropriate time to compare the best performances that one can obtain with these machines against conventional supercomputers.

Williams[1] has addressed the problem of doing unstructured mesh computations on an MIMD parallel computer. Here the solution is computed first on one node using a coarse grid which is then adaptively refined while distributing the load and the data to multiple processors as and when necessary. In practice, however, one has to compute the solution on a given grid and furthermore, it may not be feasible to generate a coarse grid for complex configurations in three dimensions which will fit on one processor. We address here the problem of computing on a given grid which does not fit in one processor. In the implementation discussed here, no hierarchy of processors is assumed. Each processor receives about the same amount of data and performs nearly the same amount of computation. The amount of communication that a processor performs is only a function of the data partitioning strategy for a given numerical scheme.

[1]Applied Research Branch, Mail Stop T045-1, Numerical Aerodynamic Simulation (NAS) Systems Division, NASA Ames Research Center, Moffett Field, CA 94035. The author is an employee of Computer Sciences Corporation. This work was funded under contract NAS 2-12961.

[2]Computational Fluid Dynamics Branch, Mail Stop 202A, NASA Ames Research Center, Moffett Field, CA 94035.

In this paper, we discuss an explicit upwind finite-volume flow solver for the Euler equations in two dimensions that is well suited for massively parallel implementation. We identify aspects of the problem which need to be examined in detail in order to obtain good performance. We discuss various ways of partitioning the grid and assess their impact on performance. We also derive efficient data structures and schedules of messages. We present detailed performance comparisons of this code on the Intel iPSC/860 against other implementations. This solver has been implemented on the Cray Y-MP by Barth and Jespersen [2] and on the CM-2 by Hammond and Barth [3].

2 Governing Equations and Discretization

The Euler equations in integral form for a control volume Ω with boundary $\partial\Omega$ read

$$\frac{d}{dt}\int_{\Omega} \boldsymbol{u}\, dv + \oint_{\partial\Omega} \boldsymbol{F}(\boldsymbol{u},\boldsymbol{n})\, dS = 0. \tag{1}$$

Here $\boldsymbol{u}$ is the solution vector (the conserved variables density, the two components of momentum and total energy). The vector $\boldsymbol{F}(\boldsymbol{u},\boldsymbol{n})$ represents the inviscid flux vector for a surface with normal vector $\boldsymbol{n}$. Eqn. (1) states that the time rate of change of the variables inside the control volume is the negative of the net efflux of the variables through the boundaries of the control volume. This net efflux through the control volume boundary is termed the residual. In the present scheme the variables are stored at the vertices of a triangular mesh. The control volumes are non-overlapping polygons which surround the vertices of the mesh. They form the "dual" of the mesh, which is composed of segments of medians. Associated with each edge of the original mesh is a (segmented) dual edge. The contour integrals in Eqn. (1) are replaced by discrete path integrals over the edges of the control volume. Fig. 1 illustrates a sample triangulation. The shaded region is the control volume associated with vertex P. A piecewise linear variation is assumed within each control volume. In order to compute this variation, the values as well as the gradients of the variables at the vertices are required. The gradients are also computed as discrete path integrals over the edges of the control volume by making use of Green's theorem. The formation of the residual at a vertex then involves information consisting of values and gradients from its distance 1 (immediate) neighbors, and the gradients at the neighbors in turn utilize values from their distance 1 neighbors. Thus, a vertex is influenced by its distance 1 and distance 2 neighbors. A distance 1 neighbor of a vertex is defined as a vertex which is connected to the chosen vertex by an edge of the triangular mesh. Vertices i and j are distance 2 neighbors if the shortest path between them comprises of two edges. Fig. 1 also illustrates the stencil associated with a vertex which consists of both distance 1 and distance 2 neighbors.

A four stage Runge-Kutta scheme is used to advance the solution in time. The path integrals are evaluated by looping over the edges of the original mesh. For the parallel implementation we assume that the triangulation of the computational domain and the mesh connectivity information are provided.

3 MIMD Implementation

In this section we examine in detail a number of issues that are crucial to obtaining good performance with the unstructured grid solver. The main steps involved are:

- partitioning of the grid
- generating local data structures

- embedding onto the hypercube topology
- generating a communication schedule

The static partitioning and the transformation of the global data set into local data sets are currently done as pre-processing steps on a workstation.

3.1 Partitioning of the grid

In this subsection we address the problem of partitioning unstructured triangular meshes. When using piecewise linear reconstruction, the calculation of the residual at a vertex vertex depends on information not only from its distance 1 neighbors, but also its distance 2 neighbors (Fig. 1). Thus, at first glance, it would appear that we need an overlap between the sub-domains. Recalling the discussion in the previous section, a vertex receives information from its distance 1 neighbors in two stages: calculation of the gradients and evaluation of the path integrals in forming the residual. Therefore, abutting sub-domains (with no overlap) will suffice. We also choose to partition the triangles (as opposed to vertices), so that the inter-partition boundaries consist of edges of the original triangular mesh. The reason for this choice is discussed later in the text. In Fig. 1, an inter-partition boundary is shown by a thick line. The inter-partition boundary edges and the corresponding inter-partition boundary vertices (IBVs) are thus duplicated. Two communication phases are then required at each stage of the four stage Runge-Kutta time integration, one during the computation of the gradients and the other, during the formation of the residuals.

Since we are interested in partitioning triangular grids, we consider the centroidal dual mesh as the undirected graph for the partitioning problem, which is composed of lines joining the centroids of the triangles. Fig. 2a shows a triangulation and Fig. 2b shows the corresponding centroidal dual; each vertex in Fig.2b corresponds to a triangle in Fig. 2a. Following Kernighan and Lin [4], the graph partitioning problem is defined as follows. Let $T = (V, E)$ be a graph of n nodes, with sizes (weights) $w_i > 0, i = 1, ..., n$. Let $C = (c_{ij}), i, j = 1, ..., n$ be a weighted connectivity matrix describing the edges of T. Let k be a positive integer. A *k-way partition* of T is a set of $v_1, ..., v_k$ of nonempty, pairwise disjoint subsets of T, such that $\bigcup_{i=1}^{n} v_i = T$. The *cost* of a partition is the summation of c_{ij} over all i and j such that i and j are in different subsets. In the present application, all the weights are chosen to be unity.

Partitioning is done recursively starting with the problem of dividing one domain into two almost equal sub-domains. The number of triangles in the two sub-domains differs at most by one. The amount of computation in a sub-domain is mainly a function of the total number of edges, which is linearly related to the number of triangles and the number of vertices by Euler's formula for planar graphs [5]. Therefore, computational load balance is assured by this strategy if the perimeters of the sub-domain boundaries are uniform across all partitions. The various partitioning strategies only differ in the way one domain is divided into two sub-domains. Below we present the key features of three partitioning algorithms that have been employed. Further details may be found in Simon [6].

The first of the three techniques, termed coordinate bisection, makes use of the coordinate information associated with each vertex in the centroidal dual. The coordinates are then sorted in a particular coordinate direction (either x or y); one half of the ordered vertices define the first partition and the remaining vertices define the second partition. The second technique derives the partitions from the Reverse Cuthill-McKee (RCM) algorithm, which is a popular reordering technique in the direct solution of sparse matrices. This algorithm has a wavefront property and defines level sets, which represent the neighbor lists starting with a root, neighbors of the root, neighbors of neighbors of the root etc. The two partitions are defined when one half of the domain

has been traversed. The third technique is based on the spectral partitioning algorithm of Pothen et al.[7] . Their algorithm induces the partitions from the eigenvector corresponding to the second smallest eigenvalue of the Laplacian matrix associated with the graph. They have shown that the separators produced by this algorithm are shorter than those produced by RCM and nested dissection. A separator is a set of nodes which subdivides the original connected graph into two disjoint subgraphs. Separators control the fill-in that occurs during the factorization of sparse matrices, but in the present context, separators are of interest since they form the inter-partition boundaries. Simon [6] has applied these three partitioning algorithms to a variety of two-dimensional and three-dimensional problems, arising from finite elements and has shown that the spectral technique yields better partitions in that it produces sub-domains with shorter boundaries. He has observed that the coordinate bisection technique leads to disconnected partitions, thereby greatly increasing the lengths of the boundary segments. Disconnected partitions also have the undesirable effect of increasing the number of adjacent partitions and each adjacent partition requires a message to be generated. Therefore, disconnected partitions imply higher start-up and transmission costs. The RCM technique produces partitions with long boundaries since it uses a breadth-first search to define the level sets. The spectral technique produces uniform, mostly connected sub-domains with short boundaries. Theoretical results by Fiedler (summarized in [7]) show that one of the two sub-domains formed by the spectral partitioning is always connected. Spectral partitioning results in fewer shorter messages and reduced communication cost.

We have two variants of the partitioning strategies based on each of the three techniques outlined above. The two variants are referred to as domainwise and stripwise decompositions. These are most easily explained by referring to the coordinate bisection strategy. In the domainwise decomposition, the direction in which the sorting of coordinates is performed is switched to be the longer of the coordinate directions during the recursive procedure. This technique is identical to the orthogonal bisection of Williams [1]. In the stripwise decomposition, this direction is fixed throughout the partitioning procedure to be the longer of the two coordinate directions in the original graph. The domainwise decomposition produces partitions which have more neighboring sub-domains and shorter boundaries and the stripwise decomposition produces thin, long partitions which have fewer neighboring sub-domains but longer boundaries. In the implementation of the flow solver on a parallel computer, the two variants have the obvious effects of requiring more short messages and fewer long messages, respectively. In the case of the RCM, strips are formed by just partitioning the level sets for the initial graph i.e. the RCM is not carried out recursively. Similarly, for the spectral partitioning, the strips are formed by sorting the entries of the eigenvector corresponding to the initial graph and inducing the required number of partitions.

Figs. 3a,3b and 3c show eight-way domainwise partitions and Figs. 4a,4b and 4c show stripwise decompositions for a mesh around a four element airfoil. The inter-partition boundaries are shown by the thick lines in these figures. In Fig. 3a shows an eight-way partition obtained with the domainwise coordinate bisection technique. Fig. 3b and Fig. 3c show the domainwise decompositions obtained using the RCM and the spectral partitioning strategies, respectively. We observe from Fig. 3a, that even with only eight partitions, there are instances where the sub-domains degenerate to zero thickness. This is a direct consequence of the variable density of the grid within a rectangular coordinate strip and leads to disconnected domains for larger number of partitions. In Fig. 3b, we see that the partitions produced by RCM have long boundaries and are connected, but as the number of partitions increases, we have observed that the partitions become disconnected. In Fig. 3c, we notice that the partitions produced by the spectral technique are compact and this property seems to hold even as the number of partitions is increased. Figs. 4a, 4b and 4c show the stripwise decompositions achieved using the coordinate bisection, RCM and the spectral techniques, respectively. The domainwise and stripwise decompositions are characterized and compared in the

section on performance.

Finally, we observe that that the communication occurs across the shared vertices and not edges. For instance, two sub-domains which only meet at a vertex and do not share an edge, have to communicate. Therefore, the *true communication graph*, which describes the message pattern more accurately, is not the dual shown in Fig. 2b, but that shown in Fig. 2c. This graph consists of cliques corresponding to the degree of each vertex. We employ the RCM and the spectral partitioning strategies discussed above to this graph as well. This graph is dense and makes the partitioning schemes more computationally intensive. Note that the coordinate bisection only makes use of the physical coordinates and hence is not affected by the choice of the dual graph.

The execution times for the coordinate bisection, RCM and spectral techniques for a mesh with 15606 vertices on a Silicon Graphics workstation (Iris 4D/70) are 4 seconds, 3 seconds and 1750 seconds. The spectral technique is thus quite expensive. However, on a vector computer such as the Cray Y-MP, the performance of the spectral technique improves considerably because matrix vector products can be vectorized. Carrying out a number of flow simulations on the same grid also amortizes the preprocessing cost, since the partitioning needs to be done just once.

We conclude this subsection by observing that the optimal partitioning of the triangular mesh which reduces the communication time is a difficult problem. The communication time for sending a message is a function of the start-up cost and the message length. Therefore, the total communication time is governed by the start-up costs (equal to the number of adjacent partitions) and the transmission costs (proportional to the number of IBVs). The domainwise partitioning strategies attempt to reduce the transmission costs and the stripwise partitioning strategies attempt to reduce the start-up costs. To achieve truly optimal partitions, the timing model (given later) has to be included in the partitioning procedure. It is beyond the scope of this paper to address this problem.

3.2 Data structures

The information required for communication at the inter-partition boundaries is precomputed using sparse matrix data structures. Neighboring subgrids communicate to each other only through their IBVs (defined earlier) which are shared by the processors containing the neighboring subgrids. In the serial version of the scheme, field quantities (mass, momentum and energy) are initialized and updated at each vertex of the triangular grid by computing the residuals. In the parallel implementation, each processor performs the same calculations on a subgrid as it would do on the whole grid in the case of a serial computation. The difference is that now each subgrid may contain both physical boundary edges and inter-partition boundary edges, which have resulted from grid partitioning. The communication at the inter-partition boundaries consists of summing the local contributions to integrals such as volumes, fluxes, gradients etc. The data structures facilitate communication across the inter-partition boundaries in a general manner. The data structures for each processor consist of:

nadjproc - no. of adjacent processors

iadjproc - list of adjacent processors - length nadjproc

ibv - pointers to the cumulative number of IBVs in common with the adjacent processors - length nadjproc+1

nbv - number of boundary vertices in common with processor iadjproc(i). This can be derived from ibv and is not stored. nbv(j) = ibv(j+1) - ibv(j)

nintbv(.,1) - Local indices on current processor - length ibv(nadjproc+1)-1

nintbv(.,2) - Local indices on adjacent processor - length ibv(nadjproc+1)-1

We illustrate these data structures by referring to Fig. 5 which shows an eight-way partition of a domain. Only the IBVs and the shared edges are shown for the sake of clarity. They comprise the inter-partition boundaries. Each processor contains only the data that it needs for communication. The arrays take on the following values for processor 0:

nadjproc - 7

iadjproc - 1, 6, 7, 2, 4, 3, 5

ibv - 1, 7, 12, 13, 14, 15, 21, 25

nbv - 6, 5, 1, 1, 1, 6, 4

Note that sub-domains are adjacent even if they share only one vertex and no edges e.g. sub-domains 0 and 4.

After the application of the partitioning algorithms, the whole finite volume grid made up of triangular cells is partitioned into a number of subgrids equal to the number of processors. Each subgrid contains an almost equal number of triangular cells which may or may not form a single connected region. After partitioning, global values of the data structures required to define the unstructured grid are given local values within each partition. We thus dispense with any references to global indices. Additionally, each local data set contains the information a partition requires for communication at its inter-partition boundaries. Each subgrid is assigned to one processor. Below, we argue that the assignment of partitions to processors is not a crucial issue in the case of the Intel iPSC/860.

3.3 Embedding onto the iPSC/860

An *embedding* of a graph G onto the a graph H is a one-one assignment of a vertex in G to a vertex in H. We have utilized two embedding schemes, one of which is a random embedding and the other is a naive embedding wherein partition 0 is assigned to processor 0, partition 1 is assigned to processor 1 and so on. There is little observable difference in performance between the two assignments. We define a *partition communication graph* as an undirected graph, where every edge represents the communication link between two neighboring partitions. Fig. 6 shows the partition communication graph corresponding to the decomposition shown in Fig. 5.

A near-optimal assignment of partitions to processors is an embedding of this graph G onto the hypercube graph H that minimizes the dilation cost, which is defined as the the maximum distance in H between the images of vertices that are adjacent in G [8]. The maximum degree of a vertex in the partition communication graph is arbitrary and is generally greater than the degree of a node in the hypercube. For example, it is shown in the section on performance that with spectral partitioning on the centroidal dual, we obtain a maximum degree of 12 with 64 processors. A near-optimal embedding (using heuristics) entails a dilation cost of at least 2, whereas a random assignment on average leads to a value of 3 for this graph.

The time (in μ secs.) to communicate a message of length m bytes over a distance d hops on the iPSC/860 is given by (cf. [9])

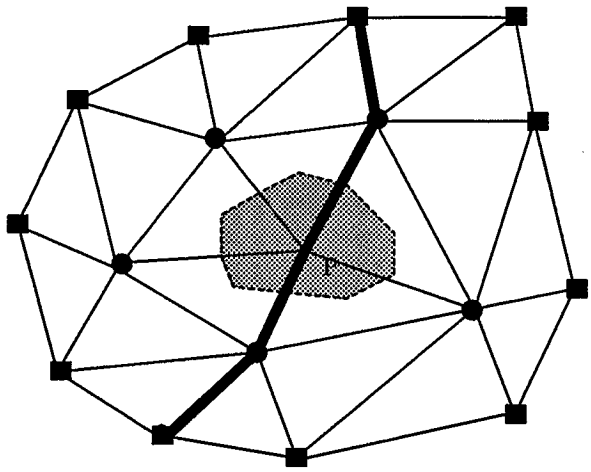
$$\begin{aligned} t &= 95 + 10.3d + 0.394m && \text{FORCED messages} \\ t &= 95 + 10.3d + 0.394m && \text{UNFORCED messages for } 0 < m \leq 100 \\ t &= 164 + 29.9d + 0.398m && \text{UNFORCED messages for } m > 100 \end{aligned} \qquad (2)$$

In the present two-dimensional code, if we have n vertices and P processors, the average length of an inter-partition boundary is roughly $\sqrt{n/P}$. In the present implementation, the gradient calculation for instance, requires 72 bytes of data per IBV, so that m assumes a value of $72\sqrt{n/P}$. In our large problem, $n = 15606$ and $P = 64$ and m is roughly 1124 bytes. It is easy to verify that even for this problem on 64 processors, a near-optimal embedding, with a dilation cost of 2, yields only an improvement in communication time of 2% over the random embedding using FORCED message type. Since the communication time is 33% of the whole computation for this problem (see the Results section), the overall improvement is less than 1%. Thus, for even relatively short messages, a near-optimal embedding only yields a small improvement over a random assignment of partitions to processors. Therefore, we have not utilized other embedding strategies, such as Gray coding after coordinate bisection etc. since we expect the performance gain to be minimal.

3.4 Communication

We now address the issue of efficient communication. We derive a scheduling algorithm so that the messages which need to be sent are scheduled with minimum conflict. The schedule we derive is most easily explained by referring to Fig. 6. The scheduling algorithm consists of coloring the edges of the partition communication graph. This algorithm permutes the adjacency sub-domain indices (iadjproc) associated with each processor to produce "pairwise exchanges". The coloring of the edges has the property that the number of colors is equal to either the maximum degree of a vertex or the maximum degree plus 1. Therefore, the number of "stages" taken for the entire communication is equal to the number of messages (+1) required by the processor that has to communicate the most. A "stage" here is defined as the step in the communication when processor pairs can communicate without interference from other processors. We show in Table 1 the original adjacency processor list and in Table 2, the schedule for partition communication graph of Fig. 6. Each column in Table 2 represents a stage. In each column, the processors are so arranged that they can communicate exclusively with one of their neighbors. In the fourth column, for example, processors 0 and 2, 1 and 7, 3 and 5 communicate, while processors 4 and 6 idle. We do not enforce explicit synchronizations at the end of every stage, so that the stages shown in Table 2 are for the purpose of clarity only. We can omit the schedule completely and post asynchronous receives (IRECV) for all the messages that a processor expects to receive and provide buffers for each of these messages. However, this approach leads to a considerable waste of memory, does not utilize the concurrent communication facility of iPSC/860 and thus limits performance. Organizing the messages into pairs also enables us to utilize the concurrent bidirectional communication on the Intel iPSC/860. Following Seidel et al. [10], this is done by the synchronization of a pair of processors which is accomplished by exchanging zero byte messages.

Two qualifications need to be made regarding the optimality of the schedule described above. First, the schedule does not guarantee the absence of wire contentions at any stage of the communication in Table 2. Wire contentions inhibit the concurrency of the scheduled messages and lead to sub-optimal communication times. The Intel iPSC/860 uses the the e-cube routing algorithm for communication between two processors ([9]), so that the path between two processors is deterministic. Schedules have been derived for simpler communication patterns, such as a complete exchange, which avoid link contentions (see [9] and [10]), but are quite difficult to derive for general graphs. Second, the schedule is optimal only under the assumption that the start-up cost dominates the transmission cost or that the message lengths are uniform across all the processors. When the message lengths are disparate, the schedule does not ensure that the communication time is minimized over all possible schedules. This is a difficult problem since we have to account for delays due to varying message lengths in the schedule and we do not it pursue it further.



● - Distance 1 (immediate) neighbors of vertex P

■ - Distance 2 neighbors of vertex P

Figure 1. Control volume and stencil for vertex P.

Table 1: **Adjacency processor list for the sample problem.**

Processor	iadjproc						
0	1	6	7	2	4	3	5
1	2	0	7	3	6		
2	3	1	0	4	7		
3	4	2	0	5	1		
4	5	3	0	2			
5	0	3	4	6			
6	5	7	0	1			
7	1	0	6	2			

Table 2: **Communication Schedule.**

Processor	Permuted iadjproc						
0	1	6	7	2	4	3	5
1	0	2	3	7	6		
2	3	1	4	0	7		
3	2	4	1	5	-	0	
4	5	3	2	-	0		
5	4	-	6	3	-	-	0
6	7	0	5	-	1		
7	6	-	0	1	2		

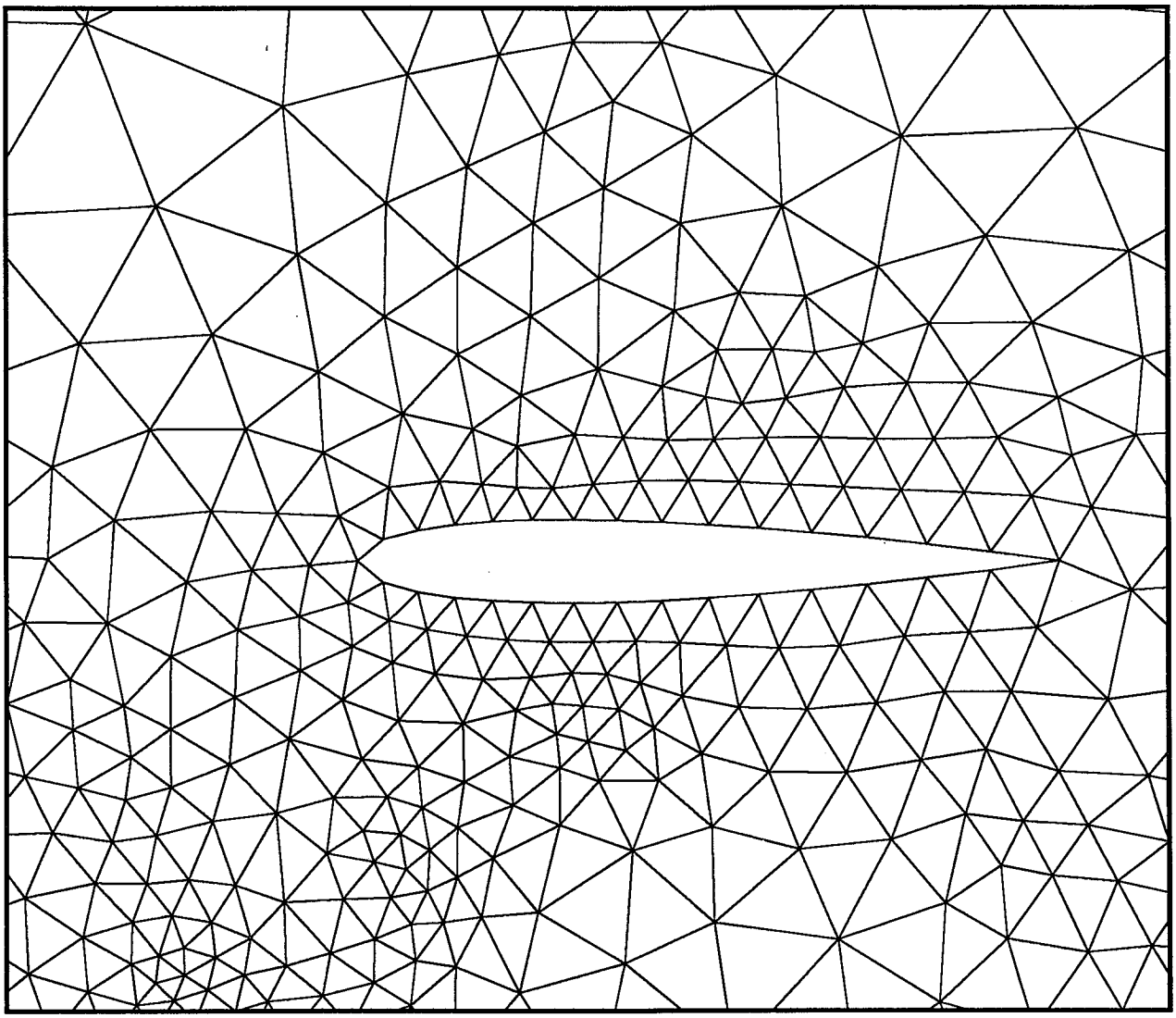


Figure 2(a). Triangulation.

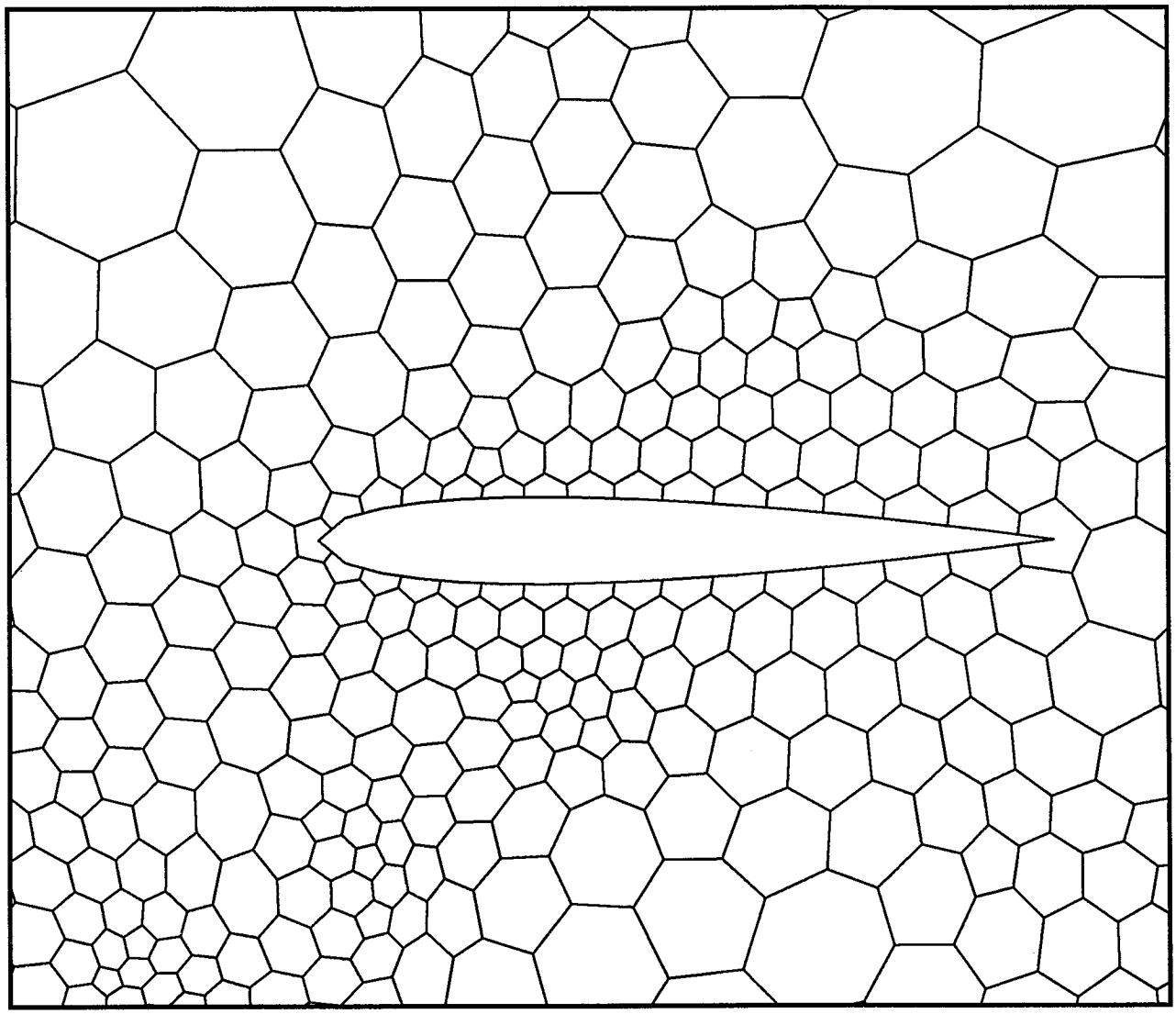


Figure 2(b). Dual grid.

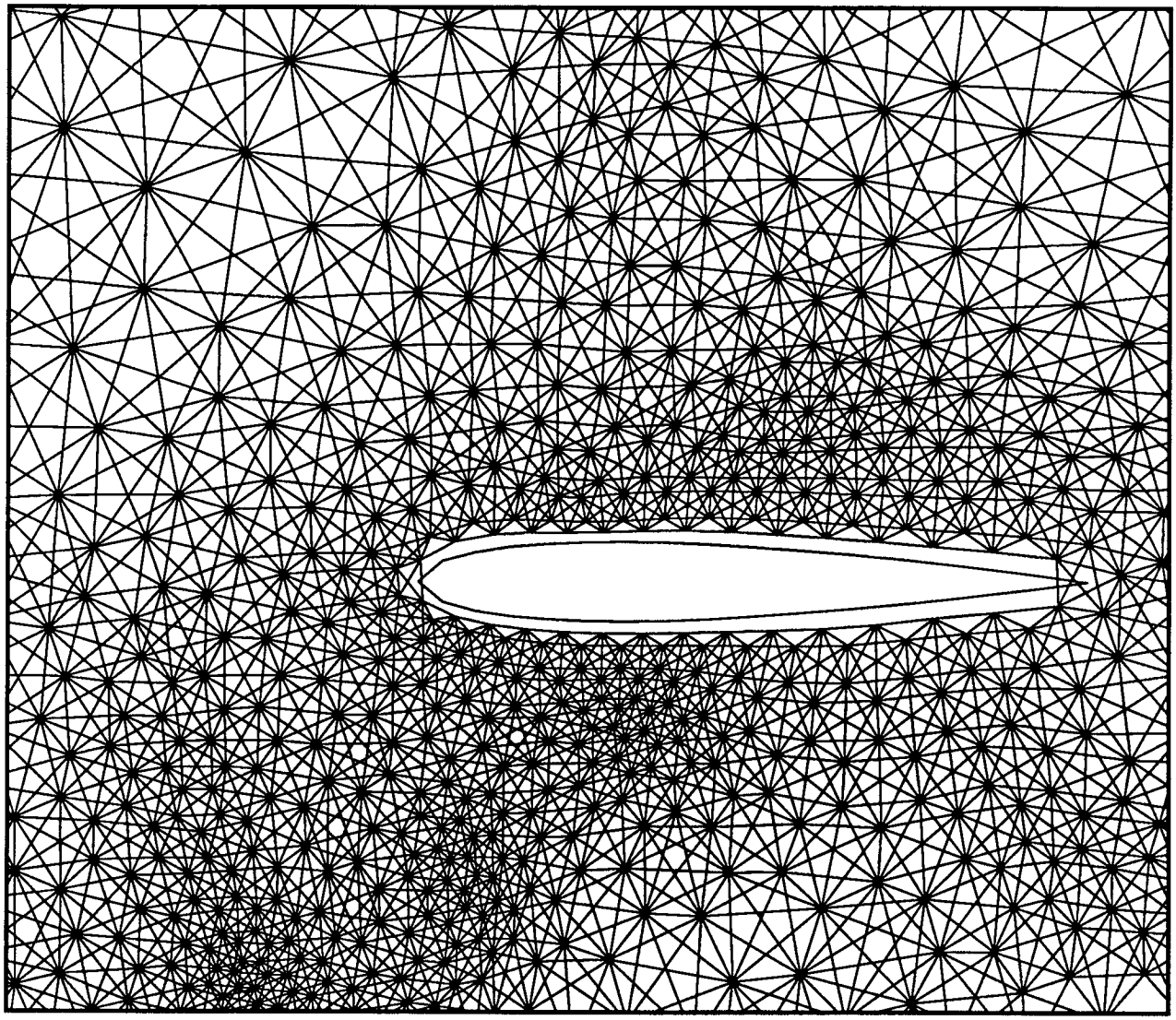


Figure 2(c). True communication graph.

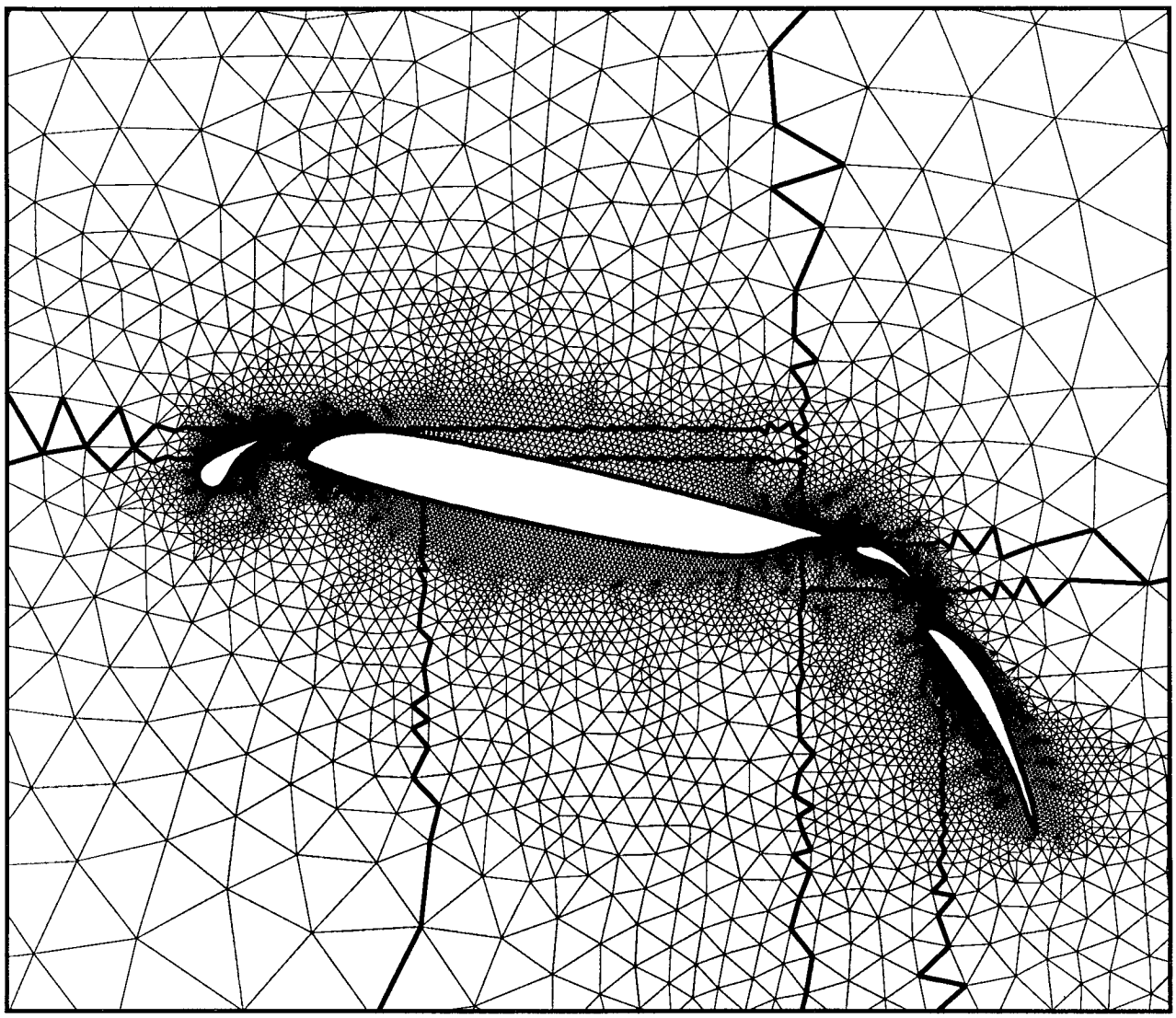


Figure 3(a). Domainwise decomposition using coordinate bisection.

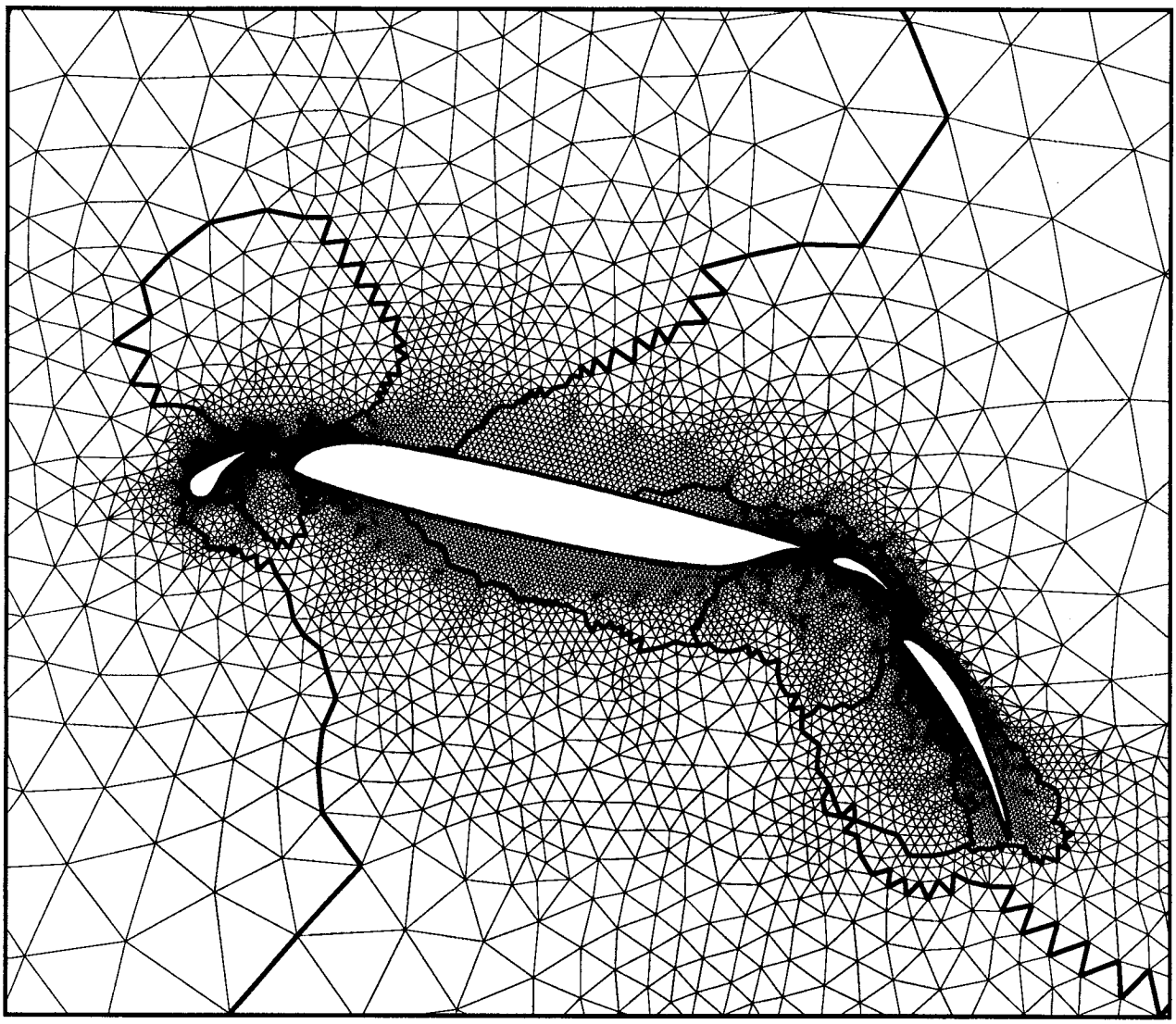


Figure 3(b). Domainwise decomposition using RCM.

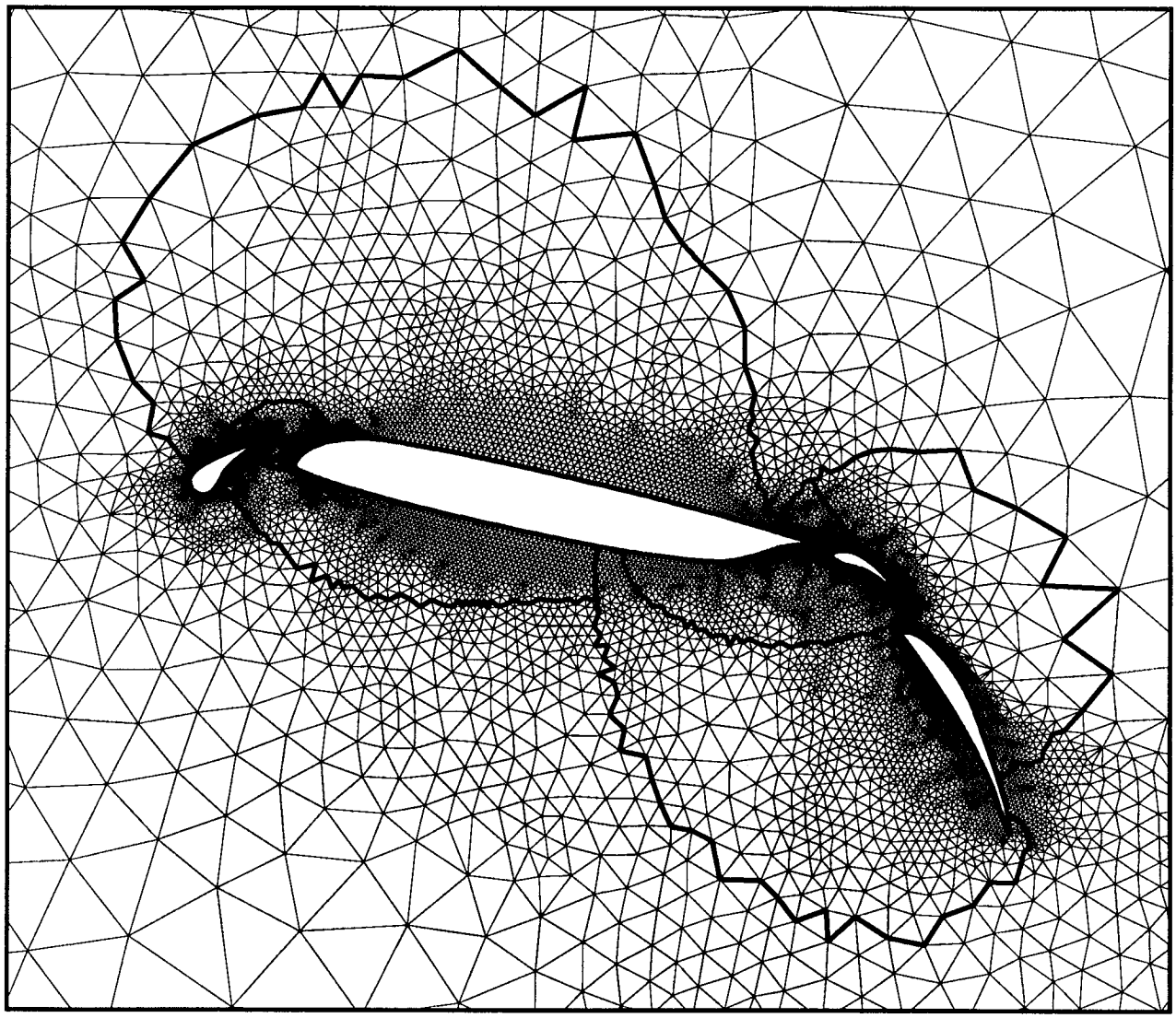


Figure 3(c). Domainwise decomposition using spectral technique.

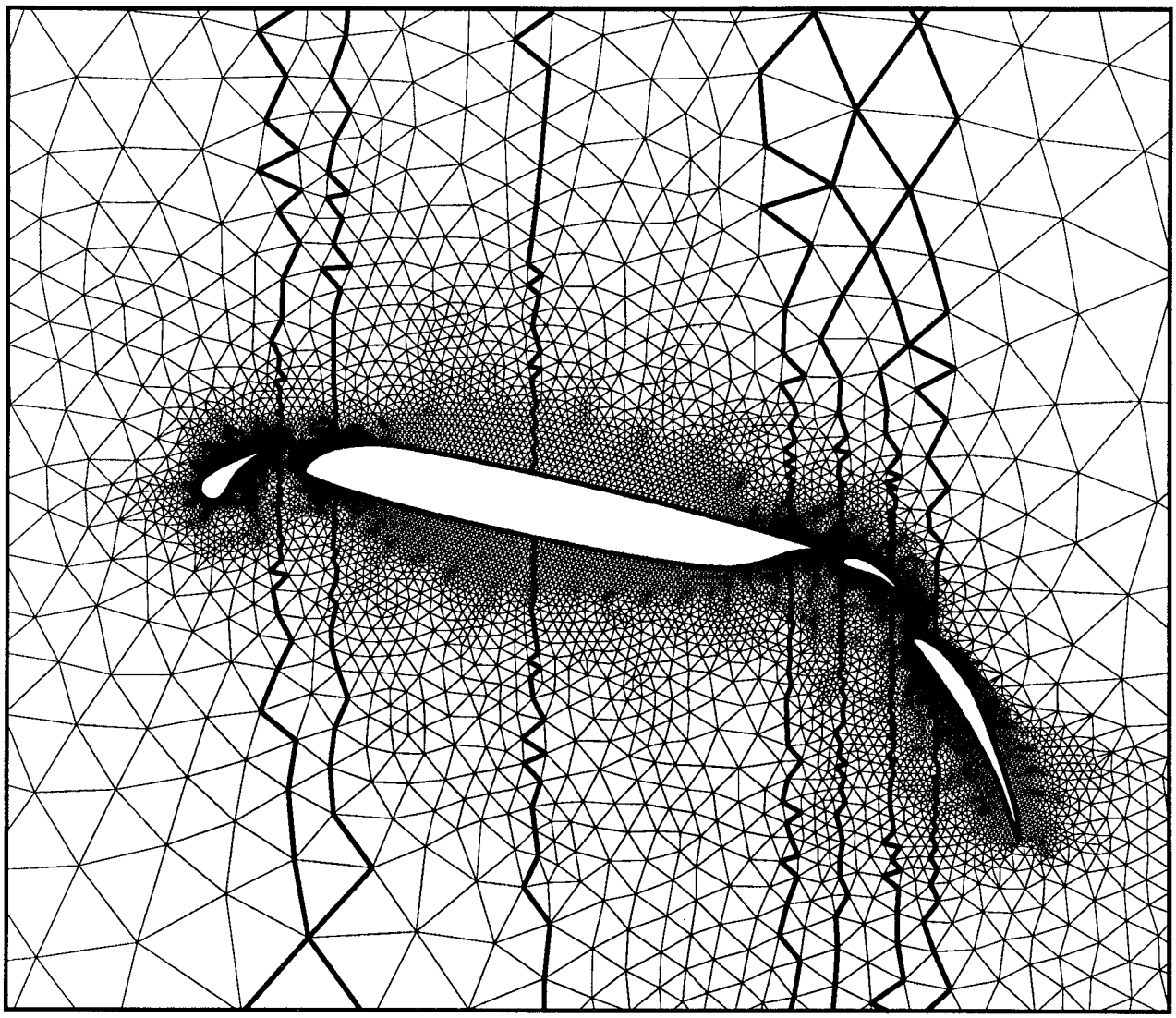


Figure 4(a). Stripwise decomposition using coordinate bisection.

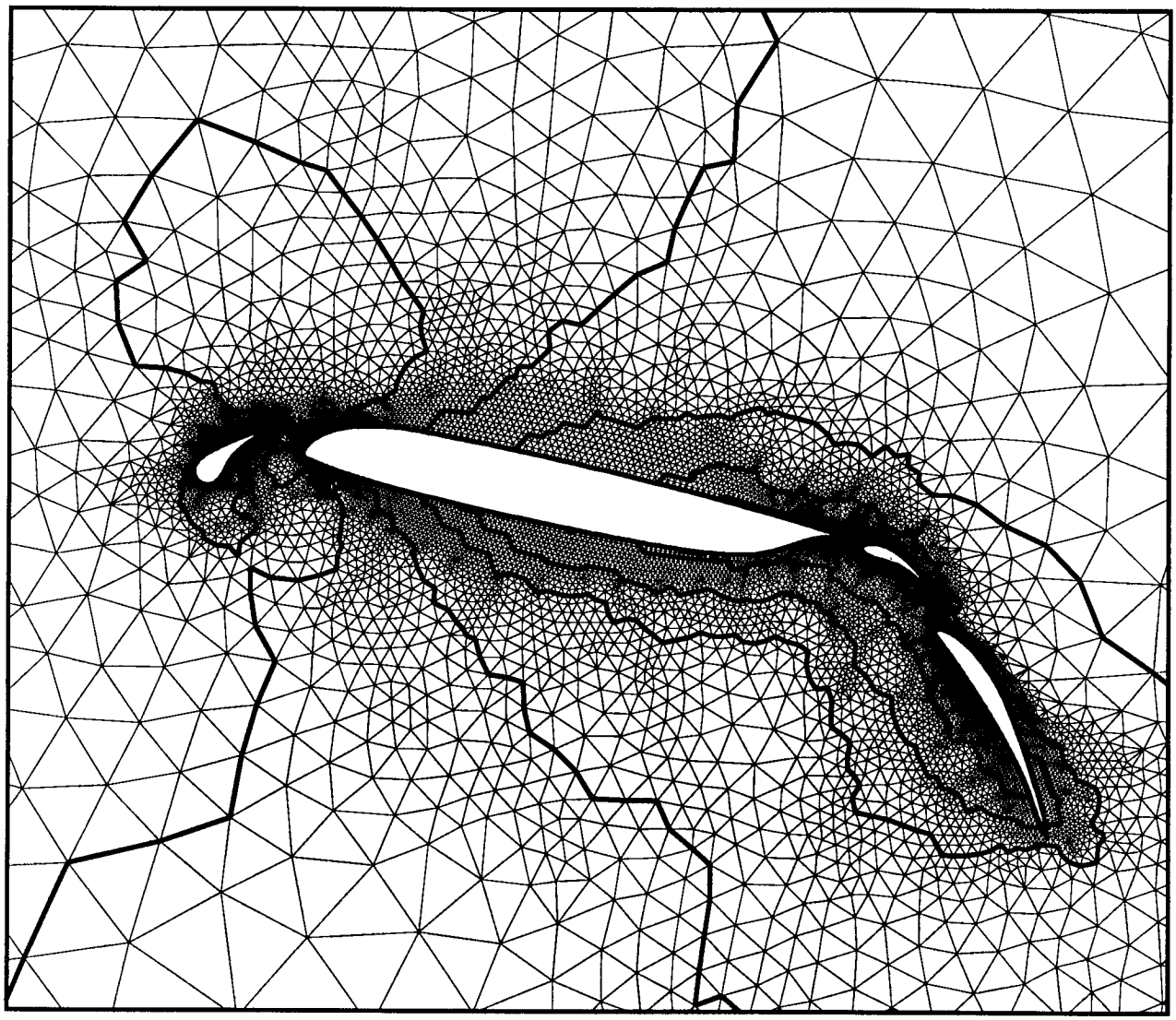


Figure 4(b). Stripwise decomposition using RCM.

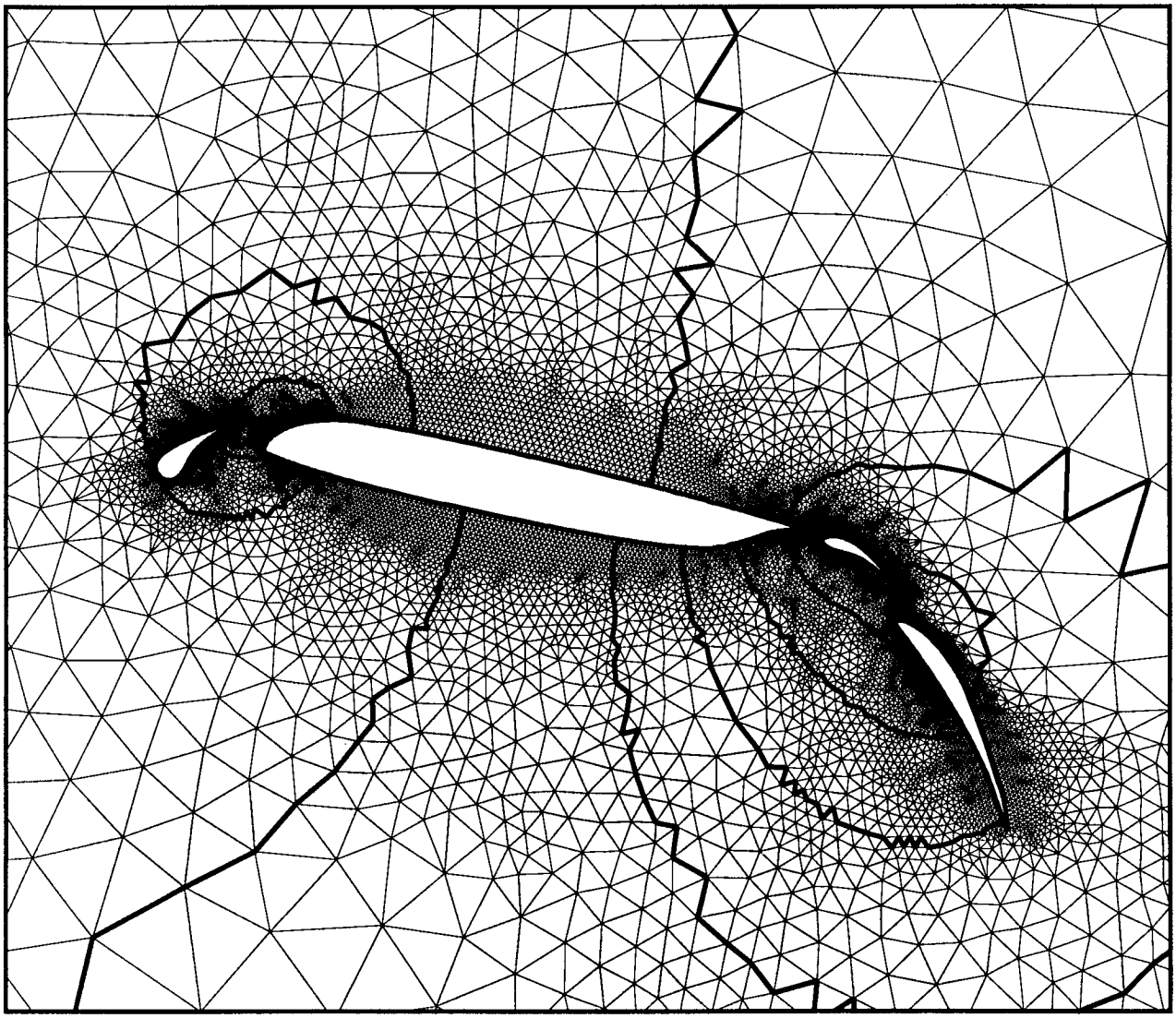


Figure 4(c). Stripwise decomposition using the spectral technique.

We now discuss the reason for partitioning triangles and not vertices. We show that there is a duality between the two choices and partitioning triangles is a better choice for the iPSC/860. The following discussion pertains to the communication required for the formation of the residual. When partitioning triangles, the inter-partition boundary consists of the edges of the triangular mesh as shown in Fig. 1 for a sample triangulation. One two-way message is required for every shared vertex in order to compute the contribution to a path integral and the total message length is directly proportional to the number of shared vertices. When partitioning vertices, the inter-partition boundary consists of the edges of the dual mesh. It is shown in Fig. 7 as a thick line for the same triangulation. Communication is required at the time of computation of the fluxes. The formation of the residual (the discrete path integral) itself requires no communication. Hammond and Barth [3] advocate the use of a vertex partitioning strategy for implementation on a Connection Machine. In their strategy, the flux computation at the inter-processor boundary is performed by one of the processors. This processor sends one message to receive the data from the neighbor, computes the flux and sends back the flux information. It thus requires two one way messages. For a fine-grained parallel computer, this approach is advantageous (since each processors is assigned one vertex). The Intel iPSC/860 is a medium grained parallel computer that is capable of bidirectional communication. If we duplicate the flux calculation at the boundary (which is not a severe penalty on a medium grained computer), the communication consists of one two-way message which can make use of the bidirectional communication on the Intel iPSC/860. However, the total message length is proportional to the number of cut edges i.e. edges with one vertex in each processor. The number of cut edges is always greater than the number of shared vertices for triangular meshes (typically twice). Therefore, partitioning triangles is better than partitioning vertices, since it entails shorter messages. In three dimensions, the number of cut edges for tetrahedral meshes is even greater (typically three times the number of shared vertices) and we expect partitioning tetrahedra to be superior to partitioning vertices.

4 Performance on the Intel iPSC/860

We consider a subcritical flow past a four element airfoil in landing configuration with a freestream Mach number $M_\infty = 0.1$, and angle of attack of 5°. Performance results are presented for two problem sizes that are representative for two-dimensional inviscid flows. The coarse mesh has 6019 vertices, 17473 edges, 11451 triangles, 4 bodies and 593 boundary edges. The fine mesh has 15606 vertices, 45878 edges, 30269 triangles, 4 bodies and 949 boundary edges. In the Cray implementation, vectorization is achieved by coloring the edges of the mesh. More details may be found in [2]. The fully vectorized code for this test case runs at 150 Mflops on a single processor of the NAS Cray Y-MP at NASA Ames, and requires 0.15 seconds per time step for the smaller grid and 0.39 seconds per time step for the larger grid. The performance of the code on the Intel iPSC/860 is given in Table 3. Table 3 represents the performance results from the best of our efforts for the two problem sizes. These results are from using the domainwise spectral partitioning based on the centroidal dual, the schedule, the bidirectional communication capabilities of the iPSC/860 and Fortran compiler optimizations. The effects of the various optimizations outlined in the previous sections are examined in detail in the following. The Mflops numbers in this section are based on operation counts using the Cray hardware performance monitor, i.e. are based on the operation count for a vectorized implementation. Since the larger problem does not fit in one processor of the iPSC/860, the efficiency is computed based on the formula:

Efficiency(%) = (Mflops with N procs) / [N * 4.6]*100 . where 4.6 Mflops is the performance of one processor of iPSC/860 on the smaller problem.

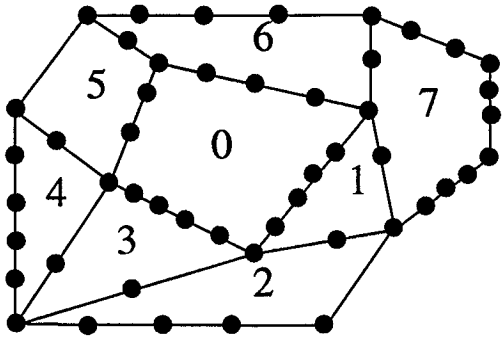


Figure 5. Decomposition into eight partitions.

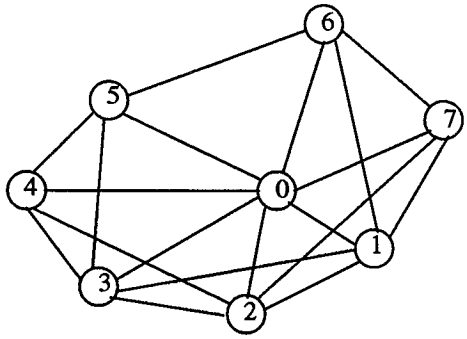


Figure 6. A Partition Communication Graph.

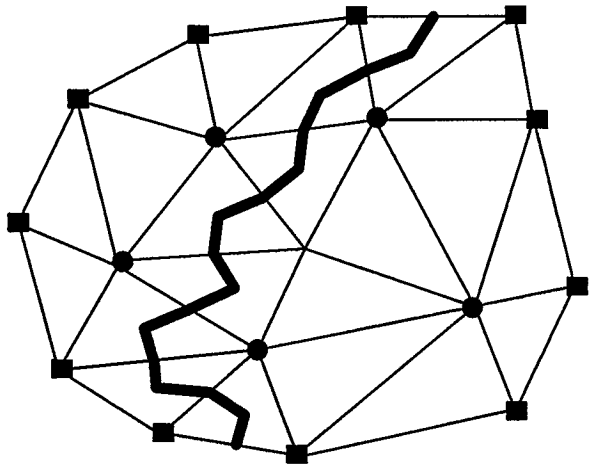


Figure 7. Inter-partition boundary for the partitioning of vertices

The numbers in Table 3 show that with 128 processors on the iPSC/860 using 64-bit arithmetic, we are able to obtain slightly more than twice the performance of a single processor of the Cray Y-MP. It also shows that the performance of the code on a single processor of the iPSC/860 is about 4.8 Mflops, considerably less that the advertised peak speed of this processor of 60 Mflops. A third point to note from Table 3 is that for a fixed problem size, the efficiency decreases as the number of processors is increased and that for the same number of processors the performance improves with increasing problem size. However neither of these trends is uniform (due to cache effects). The best performances of the code on the iPSC/860 for the two problems are shown graphically in Fig. 8. The ideal curve is determined by assuming 4.8 Mflops per processor and no communication. There is sufficient memory on the iPSC/860 to run much larger problems since two processors are enough to solve test case 2.

The various partitioning strategies are compared in Tables 4 and 5. In Table 4, the results from domainwise partitions are shown and in in Table 5, results from stripwise partitions are shown. In both tables the results derived from applying the partitioning techniques to both the centroidal dual and the true communication graph (described in section 3.1) are shown. All the results are shown for the larger problem with the number of processors fixed at 64. The tables present the total time taken per time step, the time spent in communication alone and the Mflop rates. The times shown are the maximums across all processors. We can also characterize the efficacy of the partitioning methods by the average number of neighboring partitions and the total number of shared boundary vertices (which represents the perimeter of the sub-domain boundaries). It is seen that the spectral partitioning based on the centroidal dual produces the shortest perimeter of the sub-domain boundaries. The communication time is determined by the processor that has to communicate the most. Therefore, we also characterize the partitionings by two other measures, the maximum over all partitions of the number of adjacent partitions (nadjproc) and of the total number of IBVs that a partition has. These two measures represent the maximum over all processors of the total number of messages that need to be sent and the total length of the messages, respectively. Recall that IBVs are duplicated if shared by more than two processors. Spectral partitioning requires fewer, shorter messages as compared to the coordinate bisection and RCM strategies. Table 4 also shows that there is little to be gained from using the true communication graph for domainwise partitioning. The table shows that the choice of the partitioning scheme has a considerable impact on the communication time, and thus in the effective Mflop rating.

We also attempted to reduce latency by using stripwise decompositions which try to reduce the number of neighboring sub-domains and consequently, the message startup costs. With the spectral and the RCM techniques we use the true communication graph as well as the centroidal dual to achieve stripwise decompositions. We expect the use of the true communication graph to be crucial in reducing the number of neighbors and Table 5 seems to confirm this. In examining Table 5, we notice that both the average number of neighbors and the maximum number of neighbors do decrease, especially with the true communication graph, but at the expense of vastly increased message lengths. Using the RCM partitioning strategy with the true communication graph leads to a maximum of only 4 neighbors, but the total message length has increased more than five fold. The result is that the communication time is about four times what we achieve using domainwise spectral partitioning. We also observe in Table 5 that using the true communication graph with stripwise spectral decomposition also leads to poor communication times. We conclude that on the iPSC/860 it is still desirable to reduce the number of startups, but in such a way that the perimeters of the boundaries are not greatly increased. In examining Tables 4 and 5 we also observe that some of the schemes exhibit differing computation times (differences between total and communication times). This is mainly due to the computational load imbalance caused by having disparate sub-domain boundary perimeters. Since the various partitions have nearly the

Table 3: **Performance of Unstructured Grid Code on the Intel iPSC/860.**

Procs	small grid			large grid		
	secs/step	Mflops	efficiency	secs/step	Mflops	efficiency
1	4.86	4.6	100	-	-	-
2	3.05	7.4	80	7.39	7.9	86
4	1.31	17.1	93	3.70	15.8	86
8	0.71	31.6	86	1.94	30.2	82
16	0.41	54.9	74	1.08	54.1	74
32	0.25	90.0	61	0.59	99.2	67
64	0.17	132.4	45	0.31	187.5	64
128	0.13	173.1	29	0.19	307.9	52
Y-MP	0.15	150	-	0.39	150.0	-

Table 4: **Performance of domainwise partitioning algorithms.**
No. of vertices = 15606, No. of processors = 64

	Centroidal dual			True comm. graph	
	Coordinate	RCM	Spectral	RCM	Spectral
Seconds/time step	0.39	0.43	0.31	0.37	0.33
Mflops	150	136	189	158	177
Comm. time (secs.)	0.15	0.19	0.08	.13	0.10
Average # of neighbors	6.7	7.4	4.5	6.7	4.5
Total # of shared vertices	2528	3117	1766	2425	1756
Max. # of neighbors	14	18	12	15	4
Max. # of IBVs	175	255	99	147	109

Table 5: **Performance of stripwise partitioning algorithms.**
No. of vertices = 15606, No. of processors = 64

	Centroidal dual			True comm. graph	
	Coordinate	RCM	Spectral	RCM	Spectral
Seconds/time step	0.69	1.12	0.45	0.61	0.45
Mflops	85	52	130	96	130
Comm. time (secs.)	0.46	0.65	0.16	.35	0.18
Average # of neighbors	23.2	3.8	3.6	2.7	3.5
Total # of shared vertices	4997	13235	6012	12344	6101
Max. # of neighbors	35	6	6	4	6
Max. # of IBVs	390	1389	356	516	349

Table 6: **Performance Comparison across Architectures**

Machine	Processors	secs/step	Mflops
Cray Y-MP	1	0.39	150.0
Intel iPSC/860	64	0.31	187.5
	128	0.19	307.9
CM-2 (32 bit)	8192	0.45	130.0

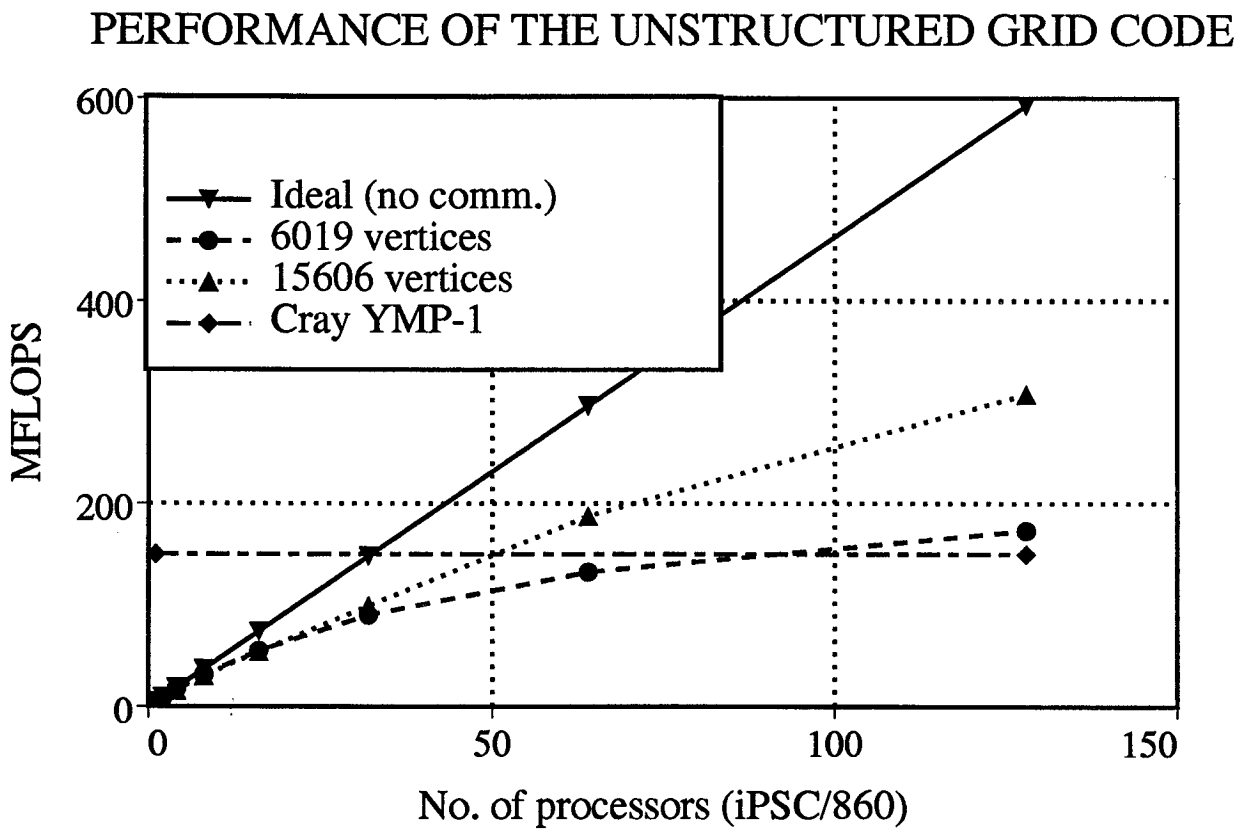


Figure 8. Parallel performance of the unstructured grid code on the iPSC/860

same number of triangles, disparate boundary perimeters lead to differing number of vertices and edges in various partitions; and these are directly related to amount of computation.

By implementing the scheduling algorithm and by forcing pairwise exchanges in a prescheduled manner, as discussed earlier, we obtained almost a factor of three improvement in performance over a naive implementation. The naive implementation here refers to having random adjacency processor lists (*iadjproc*) and using synchronized sends (CSEND) and receives (CRECV).

5 Conclusions

The performance figures for the unstructured grid code are summarized in Table 6 across three different computers, where all Mflops numbers are Cray Y-MP equivalent numbers. The CM-2 performance numbers are from [3]. The numbers in Table 6 show that current parallel supercomputers yield approximately 1-2 times the performance of a single processor Cray Y-MP for typical inviscid problems on unstructured meshes.

We have thus demonstrated in our work that an explicit unstructured flow solver can be implemented on a MIMD machine, and that supercomputer performance can be obtained. We also demonstrated the successful use of a new partitioning strategy for unstructured computations. Finally our work shows that a careful implementation of message passing is critical, even for an explicit code.

References

[1] R. D. WILLIAMS, *Supersonic fluid flow in parallel with an unstructured mesh*, Concurrency: Practice and Experience, Vol. 1, No. 1, pp. 51 – 62, September 1989.

[2] T. J. BARTH AND D. JESPERSEN, *The design and application of upwind schemes on unstructured meshes*, in Proceedings, 27th Aerospace Sciences Meeting, January 1989. Paper AIAA 89-0366.

[3] S. HAMMOND AND T. J. BARTH, *An efficient massively parallel Euler solver for two-dimensional unstructured grids*, to appear in AIAA Journal, November, 1991.

[4] B. W. KERNIGHAN AND S. LIN, *An Effective Heuristic Procedure For Partitioning Graphs*, The Bell System Technical Journal, pp. 291–308, February, 1970.

[5] J. A. BONDY AND U. S. R. MURTY, *Graph Theory with Applications*, Published by North Holland, 1982.

[6] H. D. SIMON, *Partitioning of unstructured problems for parallel processing*, Tech. Report RNR-91-08, NASA Ames Research Center, Moffett Field, CA 94035, February 1991. (to appear in Computing Systems in Engineering).

[7] A. POTHEN, H. D.SIMON, AND K.-P. LIOU, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Mat. Anal. Appl., 11 (1990), pp. 430 – 452.

[8] J. HONG, K. MELHORN, AND A. L. ROSENBERG, *Cost trade-offs in graph embeddings, with applications*, Journal of the ACM, Vol. 30, No. 4, pp. 709 – 728, October 1983.

[9] S. H. BOKHARI, *Complete exchange in the iPSC/860*, ICASE Report 91-4, ICASE, NASA Langley Research Center, Hampton, VA 23665, January 1991. (Submitted to 1991 International Conference on Parallel Processing).

[10] S. SEIDEL, M. LEE, AND S. FOTEDAR, *Concurrent bidirectional communication on the Intel iPSC/860 and iPSC/2*, Computer Science Tech. Report CS-TR 90-06, Michigan Tech. University, Houghton ,MI 49931, November 1990.